Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/110104

Honey Bee Mating Optimization with Nelder-Mead for Constrained Optimization, Integer Programming and Minimax Problems

Mohamed A. Tawhid [1,2,*] *and Kurtis Manke*[1]

[1] Department of Mathematics and Statistics, Faculty of Science, Thompson Rivers University, Kamloops, BC, Canada V2C 0C8
[2] Department of Mathematics and Computer Science, Faculty of Science, Alexandria University, Moharam Bey 21511, Alexandria, Egypt

Received: 2 Sep. 2016, Revised: 27 Oct. 2016, Accepted: 2 Nov. 2016
Published online: 1 Jan. 2017

Abstract: In this article, we propose a new hybrid Honey Bee Mating Optimization (HBMO) algorithm with simplex Nelder-Mead method in order to solve constrained optimization, integer programming and minimax problems. We call the proposed algorithm a hybrid Honey Bee Mating Optimization(HBMONM) algorithm. In the the proposed algorithm, we combine HBMO algorithm with Nelder-Mead method in order to refine the best obtained solution from the standard HBMO algorithm. We perform several experiments on a wide variety of well known test functions, 6 constrained optimization problems, 7 integer programming and 7 minimax benchmark problems. We compare the performance of HBMONM against standard HBMO algorithm and Genetic Algorithm (GA). In the majority of tests, HBMONM is shown to converge faster, and reach a better solution than both HBMO and GA in reasonable time.

Keywords: Honey bee mating optimization, Nelder-Mead method, Genetic algorithm, Constrained optimization, Minimax problem, Integer programming, Swarm intelligence.
AMS subject classification: 90C10, 90C30, 90C47, 68T20.

1 Introduction

In recent years , several biological and natural processes have been in uencing the method- ologies in science and technology in an increasing manner. Among them, a number of swarm intelligence algorithms based on the behaviour of the bees have been presented [16]. These algorithms are divided, mainly, into two categories according to their behaviour in nature , their foraging behaviour and their mating behaviour. The most well known algorithm based on the marriage behaviour of bees is the Honey Bees Mating Optimization Algorithm (HBMO) that was presented in [1], [2]and simulates the mating process of the queen of the hive. Since then, it has been used on a number of different applications [3], [6], [9], [18], [19], [20], [30]

The (HBMO) algorithm belongs to a naturally inspired branch of algorithms called swarm intelligence (SI). SI are metaheuristic algorithms that consist of a decentralized population of individuals which interact locally with one another somewhat randomly. The local interactions lead to a collective global intelligence that dictates the behavior of the population. Many SI algorithms are based on the behavior of animals or insects that tend to flock together, such as bat or ant colonies, herds of animals, and schools of fish such as the Particle Swarm Optimization (PSO) [16] and the cooperative behavior of bee colonies such as the Artificial Bee Colony (ABC) technique [14], the social foraging behavior of bacteria such as the Bacterial Foraging Optimization Algorithm (BFOA) [26], the simulation of the herding behavior of krill individuals such as the Krill Herd (KH) method [13], the mating behavior of firefly insects such as the Firefly (FF) method [32], [33] and the emulation of the lifestyle of cuckoo birds such as the Cuckoo Optimization Algorithm (COA) [27]. The HBMO algorithm aims to imitate the natural mating process of honey bees.

The goal of this work is to propose a new hybrid algorithm, namely, (HBMO) algorithm with simplex Nelder-Mead method in order to overcome the main

* Corresponding author e-mail: Mtawhid@tru.ca

drawbacks of the standard HBMO . In this paper, we propose a new hybrid algorithm, which is called simplex Honey Bee Mating Optimization (HBMONM) by combining HBMO and simplex Nelder Mead method in order to increase the exploration capability in the proposed algorithm and avoid stagnation and premature convergence in the population. Invoking the Nelder Mead method as a local search method in the final stage of the algorithm helps the proposed algorithm to accelerate the convergence and avoid performing iterations which do not imrpove the results. The HBMONM algorithm is tested on 6 constrained optimization problems, 7 integer programming and 7 minimax benchmark problems. The experimental results show that the proposed HBMONM is a promising algorithm and can obtain the optimal or near optimal solution for most of the tested function in reasonable time.

The organization of the paper is as follows. In Section **??** we present the basic algorithms such as the genetic algorithm (GA), the Nelder-Mead algorithm, and HBMO. In Section 3 we describe the proposed algorithm. In Section 4 we give the numerical experimental results for constrained optimization problems, integer programming, and minimax problems. Finally, in Section 5 we provide some concluding remarks and suggest future work.

2 The Basic Algorithms

2.1 Genetic Algorithm

The genetic algorithm (GA) is a metaheuristic algorithm that mimics natural selection and reproduction to find the global extrema. GA belongs to a larger class of evolutionary algorithms, which use biological mechanisms such as selection, reproduction, mutation, etc., to produce solutions to optimization problems. The main steps of GA are presented below:

Step 1. Randomly generate an initial population within the search space.
Step 2. Evaluate the fitness of each individual in the population.
Step 3. Choose parents according to their fitness.
Step 4. Use crossover operators on parents to produce offspring.
Step 5. Use mutation operators to alter the gene pool.
Step 6. Repeat steps 2–5 until termination criteria are met.

2.2 The Nelder-Mead algorithm

The Nelder-Mead (NM) algorithm is a derivative simplex method for finding minima for nonlinear functions [24]. The algorithm begins by creating a simplex of $n+1$ vertices $x_1, x_2, \ldots, x_{n+1}$; where n is the dimension of the problem. The function is then evaluated at each vertex, and they are ordered according to their fitness such that x_1 and x_{n+1} correspond to the best and worst vertices respectively. At each iteration new vertices are computed to form a new simplex through four operations: reflection, expansion, contraction, and shrinkage. For each operation there is a corresponding scalar coefficient defined over a range: reflection $\rho > 0$, expansion $\chi > 0$, contraction $0 < \tau < 1$, and shrinkage $0 < \phi < 1$. The algorithm of Nelder-Mead is presented in Algorithm 2.

The main steps are presented below.

The Initial Simplex
Given an initial solution x, randomly generate n neighboring solutions to form the vertices of the simplex. the function of to be minimized is then evaluated at each point, and the vertices are reordered such that x_1 is the best point, and x_{n+1} is the worst point. The centroid of these points $\bar{x}$ is calculated as:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \quad (1)$$

Reflection
The reflection process starts by computing the reflection point about the centroid $x_r = \bar{x} + \rho(\bar{x} - x_{n+1})$. If the reflected point $x_1 < x_r \leq x_n$, then the refected point is accepted and replaces x_{n+1}. If $x_r < x_n$, then the algorithm proceeds to expansion.

$x_1 < x_r$, then the reflected point is accepted and replaces x_{n+1}. If x_r is less than x_n, then the algorithm proceeds to expansion.

Expansion
The expansion process starts by computing the expansion point $x_e = \bar{x} + \chi(x_r - \bar{x})$. If $x_e < x_1$, x_e replaces x_{n+1}; otherwise x_r replaces x_{n+1} and the iteration terminates.

Contraction
If the reflected point $x_r > x_n$, the contraction process begins. There are two types of contractions: inside and outside. Which contraction is used depends on the comparison between x_r and x_{n+1}. If $x_n < x_r < x_{n+1}$, an outside contraction is performed according to $x_{oc} = \bar{x} + \tau(x_r - \bar{x})$. If $x_{oc} < x_r$ the contracted point is accepted and replaces the worst point. If $x_r > x_{n+1}$, then an inside contraction is performed according to $x_{ic} = \bar{x} + \tau(x_{n+1} - \bar{x})$. If $x_{ic} < x_{n+1}$ the contracted point is accepted and replaces the worst point. If either contraction is accepted, the iteration terminates.

Shrinkage
If no contraction point was accepted the shrink process executes by shrinking all points toward the best point to create a new simplex: $x_1, x_2, \ldots, x_{n+1}$, where $x_i' = x_1 + \phi(x_i - x_1)$, $i = 2, \ldots, n+1$.

At the beginning of each iteration, the vertices are reordered, and the centroid recalculated.

Algorithm 1 The NM Algorithm

1: Let x_i denote the list of vertices in the current simplex, $i = 1,\ldots,n+1$.
2: **1. Order.**
3: Order and relabel the vertices such that $f(x_1)$ and $f(x_{n+1})$ are the lowest and highest function values respectively.
4: **while** $f(x_1) - f(x_{n+1}) > tolerance$ **do**
5:   **2. Reflection.**
6:   Compute the centroid of the simplex $\bar{x} = \sum x_i/n, i = 1,\ldots,n$.
7:   Compute the reflection point $x_r = \bar{x} + \rho(\bar{x} - x_{n+1})$.
8:   **if** $f(x_1) \leq f(x_r) < f(x_n)$ **then**
9:     Replace x_{n+1} with x_r and proceed to step 6.
10:   **end if**
11:   **3. Expansion.**
12:   **if** $f(x_r) < f(x_1)$ **then**
13:     Compute the expansion point $x_e = \bar{x} + \chi(x_r - \bar{x})$.
14:     **if** $f(x_e) < f(x_r)$ **then**
15:       Replace $f(x_{n+1})$ with $f(x_e)$ and proceed to step 6.
16:     **else**
17:       Replace $f(x_{n+1})$ with $f(x_r)$ and proceed to step 6.
18:     **end if**
19:   **end if**
20:   **4. Contraction.**
21:   **if** $f_n \leq f(x_r) < f(x_{n+1})$ **then**
22:     Compute an outside contraction $x_{oc} = \bar{x} + \tau(x_r - \bar{x})$.
23:     **if** $f_{oc} \leq f(x_r)$ **then**
24:       Replace x_{n+1} with x_{oc} and proceed to step 6.
25:     **end if**
26:   **else**
27:     Compute an inside contraction $x_{ic} = \bar{x} + \tau(x_{n+1} - \bar{x})$.
28:     **if** $f(x_{ic}) \leq f(x_{n+1})$ **then**
29:       Replace x_{n+1} with x_{ic} and proceed to step 6.
30:     **end if**
31:   **end if**
32:   **5. Shrinkage.**
33:   Evaluate the n new vertices $x'_i = x_1 + \phi(x_i - x_1), i = 2,\ldots,n+1$.
34:   Replace vertices $x_2,\ldots,x_{n+1}$ with $x'_2,\ldots,x'_{n+1}$.
35:   **7. Reordering.**
36:   Order and relabel the vertices such that $f(x_1)$ and $f(x_{n+1})$ are the lowest and highest function values, respectively.
37: **end while**

2.3 Honey Bee Mating Optimization Algorithm

The main steps in the original HBMO algorithm are presented below.

The Mating Flight

At the beginning of each mating flight, the speed and energy of the queen are randomly generated. A random drone is then generated and its fitness is evaluated. A successful mating between the queen and a drone is determined probabilistically through an annealing function as follows:

$$prob(Q,D) = e^{\frac{\Delta(f)}{S(t)}}, \tag{2}$$

where $prob(Q,D)$ is the probability of the drones chromosome D being added to the spermatheca of the queen Q, $\Delta(f)$ is the difference between the fitness of the queen $f(Q)$ and the fitness of the drone $f(D)$, and $S(t)$ is the speed of the queen at time t.

A successful mating occurs if the value of $prob(Q,D)$ is greater than a randomly generated number in the range $[0,1]$. If the mating is successful, then the drone's sperm is added to the queen's spermatheca.

After each attempted mating, the queen transitions to a new randomly generated drone, and the speed and energy of the queen decay according to the equations:

$$S(t+1) = \alpha S(t), \tag{3}$$

$$E(t+1) = E(t) - \beta, \tag{4}$$

$$\beta = 0.5\frac{E(t_o)}{M}, \tag{5}$$

where α is the speed reduction variable, $E(t)$ is the energy of the queen at time t, β is the energy reduction after each transition, and M is the maximum number of mating flights.

The stopping criterion for each mating flight is reached when her spermatheca is full, or the speed or energy has reached its respective minima.

Breeding

After the mating flight is complete, random genes are selected from the queen's spermatheca and combined with the queen's genome using an intermediate crossover operator as follows:

$$x_i = q_i + a(d_i - q_i), \tag{6}$$

where x_i, q_i, and d_i are the chromosomes of the offspring, queen, and drone, and a is a scaling factor chosen uniformly at random over the interval $[-0.25, 1.25]$. $[0,1]$ is a common range for a, but in our case, the larger variable range tended to produce better results. The intermediate crossover operator was chosen due to its promising performance with unconstrained problems[15].

HBMO is dissimilar to GA in creating offspring because in GA each offspring has two definite parents. Conversely, in HBMO every brood has the queen as the mother, but does not have a single drone as a father, and can have a genome consisting of a mixture of genes from the spermatheca.

Mutation of the Broods

Once all broods are created, workers are chosen according to their fitness using roulette wheel selection. A worker will apply a mutation to a brood and the brood's fitness is reevaluated. If the mutation worsens the brood's fitness, it is rejected and the brood's genome is left unchanged. Initially, the workers have equal chance to be chosen, however, after each iteration, the workers are sorted according to the change in fitness of the broods, and the highest probability is assigned to the best worker. Gaussian, uniform, non-uniform, and boundary mutation operators are used to represent the workers.

Replacement of the Queen and choosing the Elites
The final step of the algorithm is to compare the most fit brood to the queen. If the brood is more fit than the queen, it becomes the queen for the next iteration. Otherwise, the queen remains unchanged for the next iteration. All remaining broods except for a user specified number of best broods are killed. The best broods are the elite of the population and are added to the queen's spermatheca for the next mating flight.

Algorithm 2 The HBMO Algorithm

Objective min or max $f(x)$, $x = (x_1, x_2, \ldots, x_d)$.
Randomly generate a population of n drone chromosomes with random solutions.
Find the best solution Q in the initial population.
while ($t < MaxFlights$) **do**
  Randomly generate a speed $s \in [0, MaxSpeed]$.
  Randomly generate an energy $E \in [0,1]$.
  while ($s > MinSpeed$ & $E > MinEnergy$ & spermatheca is not full) **do**
    Generate a drone D with random genes.
    Calculate mating probability $prob(Q,D)$ from equation 1.
    if $rand < prob(Q,D)$ **then**
      Add drone spermatheca.
    end if
    $s = \alpha s$
    $E = \beta E$
  end while
  for $n = 1 : numOffspring$ **do**
    for $i = 1 : d$ **do**
      Draw a from a uniform distribution in $[0,1]$.
      Choose a random drone D in the spermatheca.
      Do intermediate crossover via $x_{ni} = Q_i + a(D_i - Q_i)$.
    end for
  end for
  Evaluate the fitness of broods $X = x_{n \times i}$.
  for $m = 1 : numMutations$ **do**
    Select a random gene of a random brood.
    Select a worker (mutation operator) using roullette wheel selection.
    Apply mutation to the gene.
    Update the fitness of the brood.
    Update the fitness of the worker.
  end for
  Find the current best solution Q.
  Select elite solutions (excluding Q) and add to next flight's spermatheca.
end while

3 The Proposed HBMONM Algorithm

In this section, we present the proposed HBMONM algorithm. The parameter settings used for the tests are shown in Table 2. The main steps of the algorithm are as follows.

Step 1. An initial population is generated randomly and each solution in the population has their fitness evaluated. The best solution is chosen as the queen.
Step 2. At the beginning of the queen's mating flight, her energy and speed are generated randomly. At each step of the mating flight a drone is generated randomly and mates with the queen according to an annealing function. The mating flight ends when the queen's energy or speed have reached their minimum value, or when her spermatheca is full.
Step 3. The queen's genes are randomly combined with genes from her spermatheca using an intermediate crossover operator to create broods.
Step 4. Workers are chosen according to their fitness to mutate the genes of the broods. Any improvements are kept.
Step 5. The broods are sorted according to their fitness, and the most fit brood is compared to the queen. If the brood is more fit, it becomes the queen for the next iteration. All remaining broods except for a user specified number of best broods are killed. The best broods are 'elites' that are automatically added to the queen's spermatheca for the next mating flight.
Step 6. Steps 2 through 5 are repeated until a set number of iterations have been completed. In order to increase the efficiency of the search, the NM algorithm as outlined in Algorithm 2 will be performed until the termination criterion are met.

The pseudocode for the HBMONM algorithm is outlined in Algorithm 3.

4 Results and Discussion

Twenty test functions from various categories were used to evaluate and compare the performance of the HBMO, HBMONM, and GA algorithms. The parameters for HBMONM, and GA are listed in Tables 1, 2 respectively. The values in [15] were used as a starting point for many of the parameter settings for both algorithms, but were modified to provide the best results. For certain parameters a range of values is given, as the value that provides the best computational result may be problem dependent.

4.1 Constrained optimization problems

Constrained optimization problems appear in many science, finance, operations research and engineering

Algorithm 3 The HBMONM Algorithm

Objective min or max $f(x)$, $x = (x_1, x_2, \ldots, x_d)$.
Randomly generate a population of n drone chromosomes with random solutions.
Find the best solution Q in the initial population.
while $t < MaxFlights$ **do**
  Randomly generate a speed $s \in [0, MaxSpeed]$.
  Randomly generate an energy $E \in [0,1]$.
  while ($s > MinSpeed$ & $E > MinEnergy$ & spermatheca is not full) **do**
    Generate a drone D with random genes.
    Calculate mating probability $prob(Q,D)$ from equation 1.
    if $rand < prob(Q,D)$ **then**
      Add drone spermatheca.
    end if
    $s = \alpha s$
    $E = \beta E$
  end while
  for $n = 1 : numOffspring$ **do**
    for $i = 1 : d$ **do**
      Draw a from a uniform distribution in $[0,1]$.
      Choose a random drone D in the spermatheca.
      Do intermediate crossover via $x_{ni} = Q_i + a(D_i - Q_i)$.
    end for
  end for
  Evaluate the fitness of broods $X = x_{n \times i}$.
  for $m = 1 : numMutations$ **do**
    Select a random gene of a random brood.
    Select a worker (mutation operator) using roullette wheel selection.
    Apply mutation to the gene.
    Update the fitness of the brood.
    Update the fitness of the worker.
  end for
  Find the current best solution Q.
  Select elite solutions (excluding Q) and add to next flight's spermatheca.
end while
Rank the solutions and keep the best solution x_1.
Generate the remaining $x_2, \ldots, x_{n+1}$ vertices to be used in NM.
Apply the Nelder-Mead method, as shown in Algorithm 2, until termination criterion is met.

disciplines, such as pressure vessel design problem, welded beam design problem, reliability optimization problems, potential energy functions for protein design and so on. The general form of a constrained optimization is defined as follows:

$$\text{Minimize } f(x), x = (x_1, x_2, \cdots, x_n)^T, \tag{7}$$

Subject to

$$g_i(x) \leq 0, i = 1, \cdots, m$$
$$h_j(x) = 0, j = 1, \cdots, l$$
$$x_l \leq x_i \leq x_u$$

Table 1: Parameter settings for the HBMONM algorithm

Number of workers	4
Number of queens	1
Size of spermatheca	35–45
Number of broods	10–50
Maximum number of mating flights	10^4
Initial speed	[0, 0.5–1]
Speed reduction ratio	0.9
Minimum speed	$\frac{Speed}{1000}$
Initial energy	$[0,1]$
Minimum energy	10^{-4}
Mutation rate	0.10–0.75
Number of elites	1–5
Tolerance	10^{-7} – 10^{-3}
ρ	1
χ	2
τ	0.5
ϕ	0.5

Table 2: Parameter settings for GA.

Population size	20
Selection function	Stochastic uniform
Fitness scaling	Rank
Maximum Generation	10^5
Crossover operator	Single point

where $f(x)$ is the objective function, x is a vector of n variables, $g_i(x) \leq 0$ are inequality constraints, $h_j(x) = 0$ are equality constraints, and x_l and x_u are variables bounds.

Evolutionary algorithms (EAs) have a number of advantages to solve constrained optimization, for example easy implementation, little information requirement for the problem to be solved, reliable and robust performance, etc. Due to those advantages, EAs have been successfully and broadly applied to solve COPs [5], [22], [23]. Many researchers have proposed various of EA-based constraint-handling techniques for real-parameter optimization problems which can be grouped as [5]: (1) hybrid methods; (2) separation of objectives and constraints; (3) special representations and operators; repair algorithms; (4) repair algorithms; (5) penalty functions.

The benchmark problems that were used are:

Test Problem 1 [10]. This problem is defined by

$$F_1(x) = (x_1 - 2)^2 + (x_2 - 1)^2 \tag{8}$$

subject to

$$x_1 = 2x_2 - 1, \qquad \frac{x_1^2}{4} + x_2^2 - 1 \leq 0,$$

with

$$x_i \in [-100, 100], i = 1, 2.$$

The best known solution is $f^* = 1.3934651$.

Test Problem 2 [7]. This Problem is defined by

$$F_2(x) = (x_1-10)^3 + (x_2-20)^3, \tag{9}$$

subject to

$$100-(x_1-5)^2-(x_2-20)^3 \le 0,$$
$$(x_1-6)^2+(x_2-5)^2-82,81 \le 0,$$

$$13 \le x_1 \le 100, 0 \le x_2 \le 100.$$

The best known solution is $f^* = -6961.81381$.

Test Problem 3 [11]. This problem is defined by

$$F_3(x) = (x_1-10)^2+5(x_2-12)^2+x_3^4+3(x_4-11)^2 + 10x_5^6+7x_6^2+x_7^4-4x_6x_7-10x_6-8x_7, \tag{10}$$

subject to

$$-127+2x_1^2+3x_2^4+x_3+4x_4^2+5x_5 \le 0,$$
$$-282+7x_1+3x_2+10_x3^2+x_4-x_5 \le 0,$$
$$-196+23x_1+x_2^2+6x_6^2-8x_7 \le 0,$$
$$4x_1^2+x_2^2-3x_1x_2+2x_3^2+5x_6-11x_7 \le 0,$$

$$-10 \le x_i \le 10, \quad i=1,\ldots,7.$$

The best known solution is $f^* = 680.370057$.

Test Problem 4 [11]. This problem is defined by

$$F_4(x) = 5.3578547x_3^2+0.8356891x_1x_5+ 37.293239x_1-40792.141, \tag{11}$$

subject to

$$\begin{aligned} &0 \le 85.334407+0.0056858T_1+ \\ &\quad T_2x_1x_4-0.0022053x_3x_5 \le 92, \\ &90 \le 80.51249+0.0071317x_2x_5+ \\ &\quad 0.0029955x_1x_2+0.0021813x_3^2 \le 110, \\ &20 \le 9.300961+0.0047026x_3x_5+ \\ &\quad 0.0012547x_1x_3+0.0019085x_3x_4 \le 25, \end{aligned} \tag{12}$$

$$78 \le x_1 \le 102,\ 33 \le x_2 \le 45,,\ 27 \le x_i \le 45,\ i=3,4,5.$$

Where $T_1 = x_2x_5$ and $T_2 = 0.0006262$. The best known solution is $f^* = -30665.538$.

Test Problem 5 [11]. This problem is defined exactly as Test Problem 4, except with

$$T_1 = x_2x_3, \qquad T_2 = 0.00026.$$

The best known solution is unknown.

Test Problem 6 [21]. This problem is defined by

$$F_6(x) = 10.5x_1-7.5x_2-3.5x_3-2.5x_4- 1.5x_5-10x_6-0.5\textstyle\sum_{i=1}^5 x_i^2, \tag{13}$$

subject to

$$6x_1+3x_2+3x_3+2x_4+x_5-6.5 \le 0,$$
$$10x_1+10x_3+x_6 \le 20$$

$$0 \le x_i \le 1,\ i=1,\ldots,5,\ 0 \le x_6 \le 50$$

The best known solution is $f^* = -213.0$.

For these test problems, the non-stationary penalty function employed in (Parsopoulos and Vrahatis 2002b) was used. The penalty function is defined in (Yang et al. 1997) as,

$$f(x) = F(x) + h(t)H(x) \tag{14}$$

where $F(x)$ is the original objective function of the constrained problem; $h(t)$ is a dynamically modified penalty value, where t is the current iteration number; and $H(x)$ is a penalty factor defined as

$$H(x) = \sum_{i=1}^{m} \theta(q_i(x))q_i(x)^{\gamma(q_i(x))} \tag{15}$$

where $q_i(x)$ is a relative violated function of the problems constraints, defined as $q_i(x) = \max\{0, g_i(x)\}, i = 1,\ldots,m$, and $g_i(x)$ are the problem's constraints in the form $g_i(x) \le 0$; $\theta(q_i(x))q_i(x)$ is a multi-stage assignment function (Homaifar et al. 1994); and $\gamma(q_i(x))$ is the power of the penalty function.

The parameters for the penalty function are problem dependent, using the values that provided the best results for the algorithms. The parameters are defined as

$$\gamma(q_i(x)) = \begin{cases} 1, \text{ if } q_i(x) < 0.01, \\ 2, \text{ otherwise}, \end{cases}$$

$$\theta(q_i(x)) = \begin{cases} 10, & \text{if } q_i(x) < 0.001, \\ 20, & \text{if } 0.001 \le q_i(x) < 0.01, \\ 100, & \text{if } 0.1 \le q_i(x) < 0.1, \\ 500, & \text{otherwise}, \end{cases}$$

and

$$h(t) = \begin{cases} \sqrt{t}, & \text{for Test Problem 1}, \\ t\sqrt{t}, & \text{otherwise}. \end{cases}$$

The problems' constraints in the form $g_i(x) \le 0$ were only assumed violated if $g_i(x) > 10^{-5}$. In all test problems, HBMONM and GA were executed until 10^5 function evaluations were reached. The best feasible solution was then reported. For each test problem, 30 independent experiments were performed. The first experimental test was to compare HBMO to HBMONM for constrained problems. The results of this are reported in Table 3. Figure 1 shows several examples where HBMONM reaches a lower function value faster than HBMO. Secondly, HBMONM was compared to GA, and the results are reported in Table 4. We can conclude from tables 3 and 4 that the combination of the standard HBMO algorithm with the NM algorithm can give improved results compared to both HBMO and GA.

Table 3: The mean function value after 10^5 function evaluations for the standard HBMO and hybrid HBMONM algorithms for constrained problems. The algorithm which displayed the best performance is in bold font.

Problem	HBMO	HBMONM
$f1$	1.656	**1.413**
$f2$	-6939.203	**-6961.831**
$f3$	683.800	**680.630**
$f4$	-30658.527	**-30665.551**
$f5$	-31026.435	-31026.435
$f6$	-212.996	**-213.000**

Table 4: The mean and the best solution found in all 30 runs for the constrained optimization problems. In parentheses is the sum of the violated constraints. The algorithm which exhibited the best performance is bolded.

Problem	Method	Mean Solution (Sum V.C.)	St.D.	Best Solution
f_1	**HBMONM**	**1.413** (0.0001589)	0.0511	1.3934
	GA	2.0748 (0.000093)	0.5550	1.4746
f_2	**HBMONM**	**-6961.831** (0.0002350)	0.0072	-6961.837
	GA	-6864.167 (0.0)	19.835	-6907.161
f_3	**HBMONM**	**680.630** (0.0002237)	3.7406×10^{-6}	680.630
	GA	695.937 (0.0)	5.889	686.454
f_4	**HBMONM**	**-30665.550** (0.0)	1.2878×10^{-6}	-30665.550
	GA	-30658.530 (0.0)	4.527	-30663.714
f_5	**HBMONM**	**-31026.435** (0.0)	0.0	-31026.435
	GA	-31026.356 (0.0)	0.086	-31026.428
f_6	**HBMONM**	**-213** (0.0)	4.4919×10^{-7}	-213
	GA	-212.997 (0.0)	0.003	-213.0

4.2 Minimax problems

The general form of the minimax problem is [34]

$$\min_{x} F(x), \tag{16}$$

where

$$F(x) = \max_{i=1,\ldots,m} f_i(x), \tag{17}$$

with $f_i(x)) : S \subset \mathbb{R}^n \to \mathbb{R}, i = 1,\ldots,m$. Nonlinear programming problems of the form:

$$\min_{x} F(x),$$
$$g_i(x) \geq 0, \qquad i = 2,\ldots,m,$$

can be solved as minimax problems of the form:

$$\min_{x} \max_{1\leq i\leq m} f_i(x), \tag{18}$$

where

$$\begin{aligned} f_1(x) &= F(x), \\ f_i(x) &= F(x) - \alpha_i g_i(x), \\ \alpha_i &> 0, \end{aligned} \tag{19}$$

for $2 \leq i \leq m$. For sufficiently large values of α, it can be shown that nonlinear problems can be treated as minimax problems [4]. The benchmark problems that were used are:

Test Problem 7 [34]. This problem is defined by

$$\begin{aligned} &\min_{x} F_7(x), \\ &F_7(x) = \max\{f_i(x)\},\ i = 1,2,3, \end{aligned} \tag{20}$$

$$\begin{aligned} f_1(x) &= x_1^2 + x_2^4, \\ f_2(x) &= (2-x_1)^2 + (2-x_2)^2, \\ f_3(x) &= 2e^{(-x_1+x_2)}. \end{aligned}$$

Test Problem 8 [34]. This nonlinear programming problem can be treated as a minimax problem according to (18) and (19). This problem is defined by

$$F_8(x) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4, \tag{21}$$

$$\begin{aligned} g_2(x) &= -x_1^2 - x_2^2 - x_3^3 - x_4^2 - x_1 + x_2 - x_3 + x_4 + 8, \\ g_3(x) &= -x_1^2 - 2x_2^2 - x_3^2 - 2x_4^2 + x_1 + x_4 + 10, \\ g_4(x) &= -x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4 + 5. \end{aligned}$$

Test Problem 9 [34]. This nonlinear programming problem can be treated as a minimax problem according to (18) and (19). This problem is defined by

$$\begin{aligned} F_9(x) =&(x_1-10)^2 + 5(x_2-12)^2 + 3(x_4-11)^2 + x_3^4 \\ &+ 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7, \\ g_2(x) =& -2x_1^2 - 3x_3^4 - x_3 - 4x_4^2 - 5x_5 + 127, \\ g_3(x) =& -7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 + 282, \\ g_4(x) =& -23x_1 - x_2^2 - 6x_6^2 + 8x_7 + 196, \\ g_5(x) =& -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7. \end{aligned} \tag{22}$$

Test Problem 10 [29]. This problem is defined by

$$\begin{aligned} &\min_{x} F_{10}(x), \\ &F_{10}(x) = \max\{f_i(x)\},\ i = 1,2, \end{aligned} \tag{23}$$

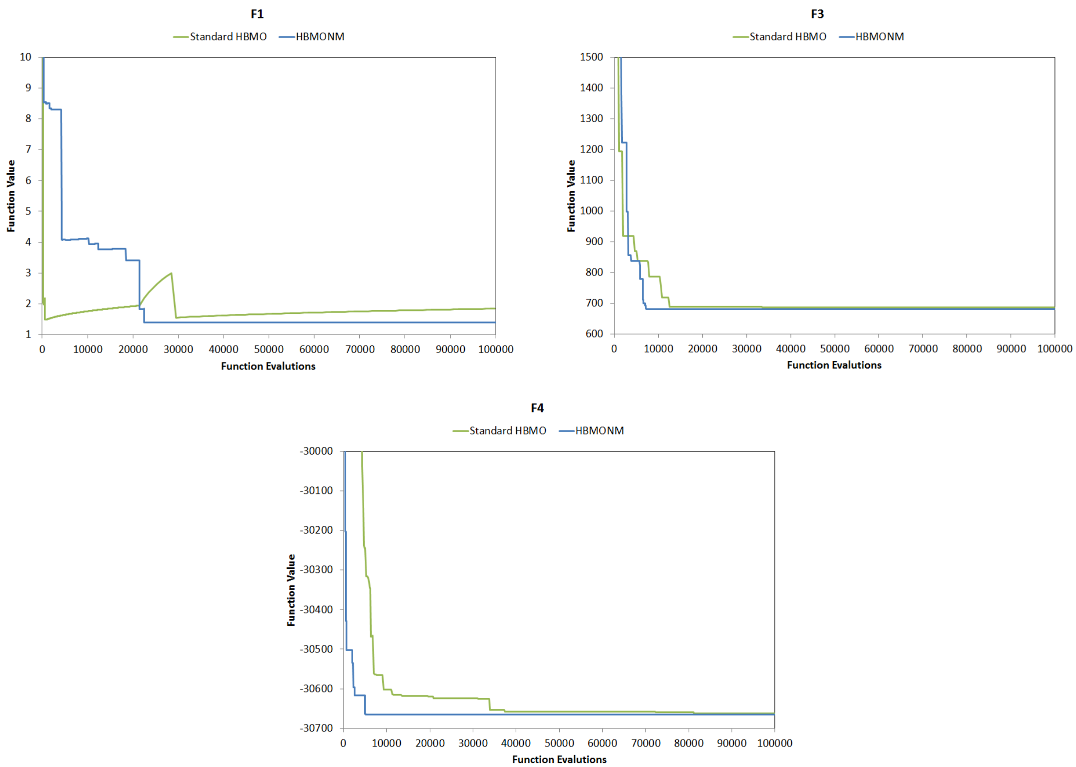

Fig. 1: The general performance of HBMONM and HBMO with constrained programming problems.

$$f_1(x) = |x_1 + 2x_2 - 7|,$$
$$f_2(x) = |2x_1 + x_2 - 5|.$$

Test Problem 11 [29]. This problem is defined by

$$\begin{aligned} &\min_x F_{11}(x) \\ &F_{11}(x) = \max\{f_i(x)\}, \\ &f_i(x) = |x_i|,\ i = 1,\dots,10. \end{aligned} \tag{24}$$

Test Problem 12 [17]. This problem is defined by

$$\begin{aligned} &\min_x F_{12}(x), \\ &F_{12}(x) = \max\{f_i(x)\}, \end{aligned} \tag{25}$$

$$f_1(x) = \left(x_1 - \sqrt{x_1^2 + x_2^2}\cos\sqrt{x_1^2 + x_2^2}\right)^2 + 0.005(x_1^2 + {}_2^2),$$

$$f_2(x) = \left(x_2 - \sqrt{x_1^2 + x_2^2}\sin\sqrt{x_1^2 + x_2^2}\right)^2 + 0.005(x_1^2 + {}_2^2).$$

Test Problem 13 [17]. This problem is defined by

$$\begin{aligned} &\min_x F_{22}(x), \\ &F_{22}(x) = \max\{|f_i(x)|\},\ i = 1,\dots,21, \end{aligned} \tag{26}$$

$$f_i(x) = x_1 e^{(x_3 t_i)} + x_2 e^{(x_4 t_i)} - \frac{1}{1+t_i},$$
$$t_i = -0.5 + \frac{i-1}{20}.$$

For each test problem, 30 independent experiments were performed with $x \in [-50, 50]^n$, where n is the dimension of the problem. An experiment was considered successful only if it reached the desired error goal within 10^{-4}, and in less than 10^5 function evaluations. HBMO was first compared to HBMONM, and the results are presented in Table 5. The results from Table 5 show that HBMONM converges much faster than HBMO, and this is also clearly shown for a several test functions in figure 2. Next HBMONM was compared to GA, and the minimum, mean, maximum, and standard deviation of the required number of function evaluations are reported in Table 6. In all cases HBMONM performs better than both HBMO and GA.

4.3 Integer programming problems

An integer programming problem is a mathematical optimization problem in which all of the variables are restricted to be integers. The unconstrained integer programming problem can be defined as follows.

$$min\, f(x),\ \ x \in S \subseteq \mathbb{Z}^n, \tag{27}$$

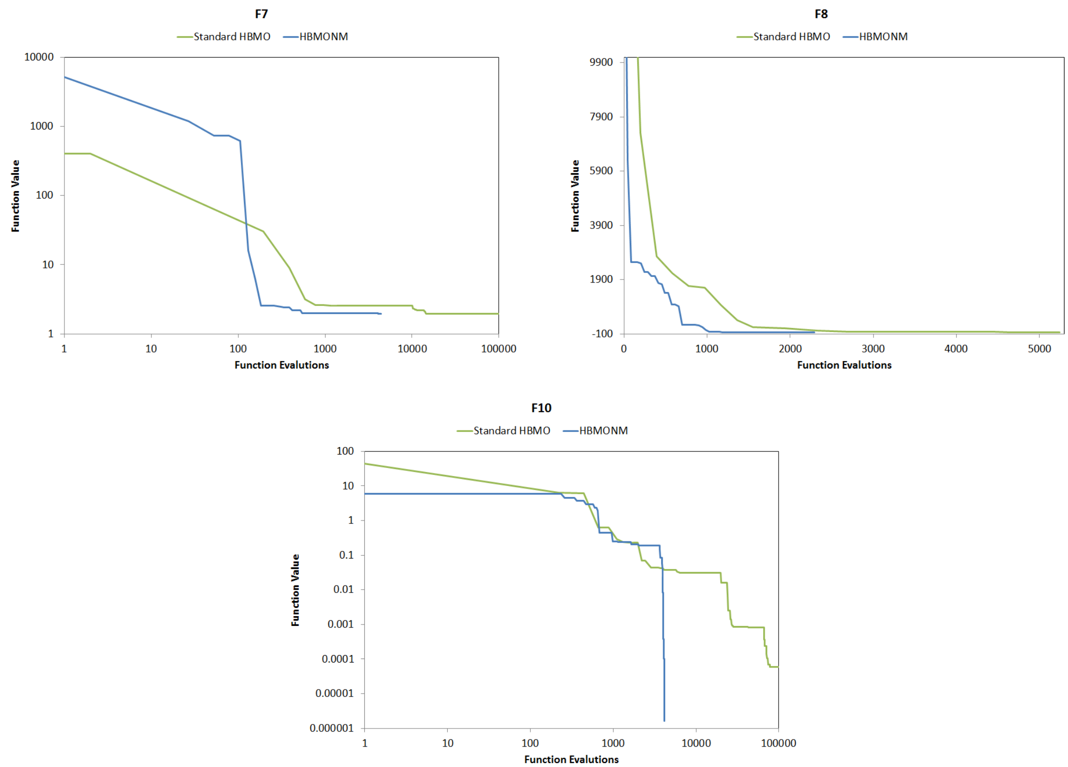

Fig. 2: The general performance of HBMONM and HBMO with minimax problems.

Table 5: The mean number of function evaluations to reach the desired error goal are shown for HBMO and HBMONM algorithms for minimax problems. If the desired error goal was not reached within 10^5 function evaluations, the experiment was stopped. The algorithm which displayed the best performance is in bold font.

Problem	HBMO	HBMONM
$f7$	39073	**658**
$f8$	78350	**3714.73**
$f9$	10^5	**6408.93**
$f10$	61358	**1067.9**
$f11$	10^5	**17225**
$f12$	58719	**19060**
$f13$	65568	**2013.7**

where $\mathbb{Z}$ is the set of integer variables, S is a not necessarily bounded set.

Now let us define the test problems.

Test Problem 14 [28]. This problem is defined by

$$F_{14}(x) = \|x\|_1 = |x_1| + \cdots + |x_n|, \tag{28}$$

where the dimension of the problem is $n = 30$. The global minimum is $F_{14}(x^*) = 0$.

Table 6: The results found in all 30 runs for the minimax optimization problems. The algorithm which exhibited the best performance is bolded.

Problem	Method	Mean	Min	Max	St.D.
f_7	**HBMONM**	**658**	99	1002	227.369
	GA	81960	75810	88040	2058.1
f_8	**HBMONM**	**3714.73**	1621	7478	1886.4
	GA	10^5	10^5	10^5	0
f_9	**HBMONM**	**6408.93**	3570	12044	240.551
	GA	10^5	10^5	10^5	0
f_{10}	**HBMONM**	**1067.9**	837	1503	153.167
	GA	87000	84375	92662	1025.741
f_{11}	**HBMONM**	**17225**	10578	22421	2635.1
	GA	10^5	10^5	10^5	0
f_{12}	**HBMONM**	**19060**	14561	25531	2550.6
	GA	38880	14260	71220	21134.7
f_{13}	**HBMONM**	**2013.7**	935	15345	2047.2
	GA	92212	90096	10^5	1434.4

Test Problem 15 [28]. This problem is defined by

$$F_{15}(x) = x^\top x = (x_1 \; \ldots \; x_n) \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \tag{29}$$

where the dimension of the problem is $n = 30$. The global minimum is $F_{15}(x^*) = 0$.

Test Problem 16 [8]. This problem is defined by

$$F_{16}(x) = -(15\ 27\ 36\ 18\ 12)x + x^{\top} \begin{pmatrix} 35 & -20 & -10 & 32 & -10 \\ -20 & 40 & -6 & -31 & 32 \\ -10 & -6 & 11 & -6 & -10 \\ 32 & -31 & -6 & 38 & -20 \\ -10 & 32 & -10 & -20 & 31 \end{pmatrix} x, \quad (30)$$

The global minimum is $F_{16}(x^*) = -737$.

Test Problem 17 [8]. This problem is defined by

$$F_{17}(x) = (9x_1^2 + 2x_2^2 - 11)^2 + (3x_1 + 4x_2^2 - 7)^2, \quad (31)$$

The global minimum is $F_{17}(x^*) = 0$.

Test Problem 18 [8]. This problem is defined by

$$F_{18}(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4, \quad (32)$$

The global minimum is $F_{18}(x^*) = 0$.

Test Problem 19 [8]. This problem is defined by

$$F_{19}(x) = 2x_1^2 + 3x_2^2 + 4x_1x_2 - 6x_1 - 3x_2, \quad (33)$$

The global minimum is $F_{19}(x^*) = -6$.

Test Problem 20 [8]. This problem is defined by

$$F_{20}(x) = -3803.84 - 138.08x_1 - 232.92x_2 + 123.08x_1^2 + 203.64x_2^2 + 182.25x_1x_2, \quad (34)$$

The global minimum is $F_{20}(x^*) = -3833.12$.

For each test problem, 30 independent experiments were performed with $x \in [-100, 100]^n$. An experiment was considered successful if the global minimum was found within 10^5 function evaluations. For both algorithms, the variables were only rounded to the nearest integer for function evaluations, and considered real numbers for all other purposes. The mean number of function evaluations required to reach the global minimum are compared for HBMO and HBMONM. The results are reported in Table 7. Next, HBMONM was compared to GA for the same set of problems, and the mean, minimum, maximum, and standard deviation of the the required function evaluations are presented in Table 8.

The results of Table 7 show that HBMONM performs better on the majority of the test problems. Comparison of the two algorithms is shown in Figure 3. Although HBMONM performed better than HBMO, Table 8 shows that GA performed better than both HBMO and HBMONM in the majority of integer programming problems.

Table 7: The mean number of function evaluations to reach the global minimum are shown for HBMO and HBMONM algorithms for integer problems. If the minimum was not reached within 10^5 function evaluations, the experiment was stopped. The algorithm which displayed the best performance is in bold font.

Problem	HBMO	HBMONM
$f14$	31318.533	**14376**
$f15$	15543.4	**13354**
$f16$	63457	**14776**
$f17$	**187.4**	652.24
$f18$	10237.6	**7454.1**
$f19$	**484.67**	525.7
$f20$	**661**	1300.6

Table 8: The mean, min, and max number of function evaluations found in all 30 runs for the integer optimization problems. The algorithm which exhibited the best performance is bolded.

Problem	Method	Mean	Min	Max	St.D.	Successes
f_{14}	**HBMONM**	**14376**	9419	34377	6716.6	30/30
	GA	33364	17700	59040	12850	30/30
f_{15}	**HBMONM**	**13354**	6178	27861	4988.5	30/30
	GA	39637.143	12160	83520	22154.3	28/30
f_{16}	HBMONM	14776	2816	53191	10565	30/30
	GA	**10474.667**	4400	22380	5354.363	30/30
f_{17}	HBMONM	652.24	62	5798	1174	30/30
	GA	**520**	180	960	238.22	30/30
f_{18}	HBMONM	7454.1	4312	19410	2953.8	30/30
	GA	**5006.67**	2480	10480	1921.37	30/30
f_{19}	**HBMONM**	**525.7**	13	2139	582.78	30/30
	GA	841.33	140	2000	496.52	30/30
f_{20}	HBMONM	1306	78	6583	1331.6	30/30
	GA	**620**	240	2000	404.84	30/30

5 Conclusion

The performance of the HBMO algorithm for constrained, minimax, and integer optimization problems was compared to a new proposed hybrid HBMONM algorithm. Through their performance on numerous widely used, well known test functions from each category, it has been shown that HBMONM consistently performs better than the standard HBMO algorithm for the majority of problems. HBMONM was then compared with GA and it was shown that HBMONM performs better than GA for all constrained and minimax problems tested, however, GA performs better on the majority of

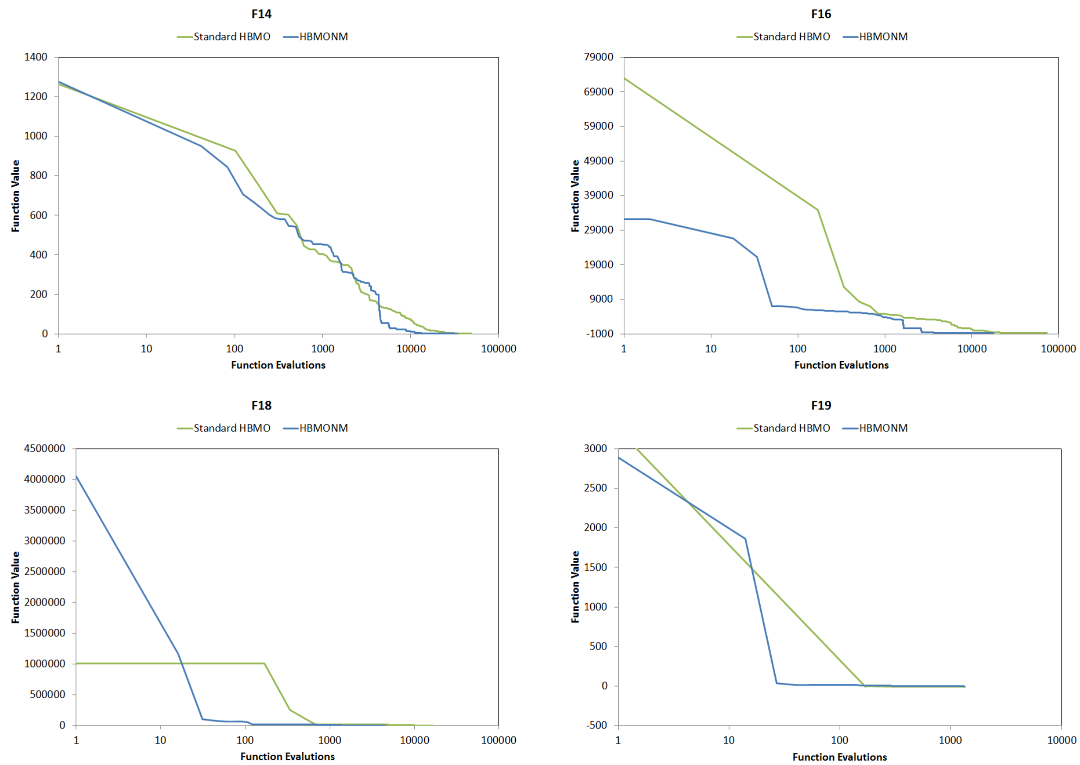

Fig. 3: The general performance of HBMONM and HBMO with integer programming problems.

integer programming problems. Further investigation may be required to improve the performance of HBMONM and HBMO on integer problems.

Acknowledgments

The research of the first author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] H. A. Abbass, A monogenous MBO approach to satisfiability. In: Proceeding of the International Conference on Computational Intelligence for Modelling, Control and Automation, CIMCA2001, Las Vegas, NV, USA, (2001).

[2] H. A. Abbass, Marriage in honey-bee optimization (MBO): A haplometrosis polygynous swarming approach. The Congress on Evolutionary Computation (CEC2001), Seoul, Korea, May 2001, 207–214 (2001).

[3] A. Afshar, O. Bozog Haddad, , M. A., Marino, B. J. Adams, Honey-bee mating optimization (HBMO) algorithm for optimal reservoir operation, Journal of the Franklin Institute, 344 (2007), 452–462.

[4] J. W. Bandler and C. Charalambous, Nonlinear programming using minimax techniques. Journal of Optimization Theory and Applications, 13 (1974), 607–619.

[5] C.A.C. Coello, Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art, Computer Methods in Applied Mechanics and Engineering, 191 (2002), 1245–1287.

[6] M. Fathian, B. Amiri and A. Maroosi, Application of honey bee mating optimization algorithm on clustering, Applied Mathematics and Computation, 190 (2007), 1502–1513.

[7] C. A. Floudas and P. M. Pardalos, A collection of test problems for constrained global optimization algorithms, In P.M. Floudas (Ed.), Lecture Notes in Computer Science, Vol. 455, Berlin: Springer (1987).

[8] A. Glankwahmdee, J. S. Liebman and G. L. Hogg, Unconstrained discrete nonlinear programming, Engineering Optimization, 4 (1979), 95–107.

[9] O. B. Haddad, A. Afshar and M. A. Marino, Honey-bees mating optimization (HBMO) algorithm: A new heuristic approach for water resources optimization. Water Resources Management, 20 (2006), 661–680

[10] D. M. Himmelblau, Applied Nonlinear Programming, New York: McGraw-Hill, (1972).

[11] W. Hock and K. Schittkowski, Test Examples for nonlinear programming codes, In Lecture notes in economics and mathematical systems, Vol. 187, Berlin: Springer, (1981).

[12] A. Homaifar, C. X. Qi and S. H. Lai, Constrained optimization via genetic algorithms, Simulation, 2 (1994), 242–254.

[13] A. Hossein and A. Hossein-Alavi, Krill herd: A new bio-inspired optimization algorithm, Communications in Nonlinear Science and Numerical Simulation, 17 (2012), 4831–4845

[14] D. Karaboga, An idea based on honey bee swarm for numerical optimization. Technical report-TR06. Engineering Faculty, Computer Engineering Department, Erciyes University, (2005).

[15] S. Karimi, N. Mostoufi and R. Sotudeh-Gharebagh, Evaluating performance of Honey Bee Mating Optimization, Journal of optimization theory and applications, 160 (2013), 1020–1026.

[16] J. Kennedy and R. Eberhart, Particle swarm optimization, In Proceedings of the IEEE international conference on neural networks, 4 (1995), 1942-?1948.

[17] L. Lukšan and J. Vlček, Test problems for nonsmooth unconstrained and linearly constrained optimization, Technical report 798, Institute of Computer Science, Academy of SCiences of teh Czech Republic, Prague, Czech Republic (2000).

[18] M. Marinaki, Y. Marinakis and C. Zopounidis, Honey bees mating optimization algorithm for financial classification problems, Applied Soft Computing, 10 (2010), 806–812.

[19] Y. Marinakis, M. Marinaki, and G. Dounias, Honey bees mating optimization algorithm for large scale vehicle routing problems, Natural Computing, 9 (2010), 5–27.

[20] Y. Marinakis, M. Marinaki and G. Dounias, Honey bees mating optimization algorithm for the Euclidean traveling salesman problem, Information Sciences, 181: 20 (2011), 4684–4698.

[21] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Berlin: Springer, (1996) .

[22] Z. Michalewicz, K. Deby, M. Schmidtz, T. Stidsenx, Test-case generator for nonlinear continuous parameter optimization techniques, IEEE Transactions on Evolutionary Computation, 4 (3) (2000), 187–215.

[23] Z. Michalewicz and M. Schoenauer, Evolutionary algorithms for constrained parameter optimization problems, Evolutionary Computation, MIT Press 4 (1), (1996), 1–32.

[24] J. A. Nelder and R. Mead, A simplex method for function minimization, Computer Journal, 7 (1965), 308–313.

[25] K. E. Parsopoulos and M. N. Vrahatis, Particle swarm optimization method for constrained optimization problems, In P. Sincak, J. Vascak, V. Kvasnicka & J. Posphical (Eds.), Intelligent technologies - theory and applications, New trends in intelligent technologies, Frontiers in technological artificial intelligence and applications, Vol. 76, pp 214–220, Amsterdam: IOS Press, (2002b).

[26] K. M. Passino, Biomimicry of bacterial foraging for distributed optimization and control, IEEE Control Systems Magazine, 22(3) (2002), 52–67.

[27] R. Rajabioun, Cuckoo optimization algorithm. Applied Soft Computing, 11 (2011), 5508–5518.

[28] G. Rüdolph, An evolutionary algorithm for integer programming, In Y. Davidor, H. -P. Schwefel & R. Männer (Eds.), Parallel Problem Solving from Nature, 3 (1994), 139–148.

[29] H. -P. Schwefel, Evolution and Optimum Seeking, New York: Wiley, (1995).

[30] J. Teo and H. A Abbass, A true annealing approach to the marriage in honey bees optimization algorithm, International Journal of Computational Intelligence and Applications, 3:2 (2003), 199–211.

[31] J.-M. Yang, Y.-P. Chen, J.-T. Horng, , and C.-Y. Kao, Applying family competition to evolution strategies for constrained optimization, In Lecture Notes in Mathematics, Vol. 1213, pp. 201–211, New York: Springer, (1997).

[32] X. S. Yang, Nature-Inspired Metaheuristic Algorithms, Luniver Press, UK, (2008).

[33] X. S. Yang, Firefly algorithms for multimodal optimisation, Proc. 5th Symposium on Stochastic Algorithms, Foundations and Applications, (Eds. O. Watanabe and T. Zeugmann), Lecture Notes in Computer Science, Vol. 5792 (2009), 169–178.

[34] S. Xu, Smoothing method for minimax problems, Computational Optimization and Applications, 20 (2001), 267–279.

Mohamed A. Tawhid got his PhD in Applied Mathematics from the University of Maryland Baltimore County, Maryland, USA. From 2000 to 2002, he was a Postdoctoral Fellow at the Faculty of Management, McGill University, Montreal, Quebec, Canada. Currently, he is a full professor at Thompson Rivers University. His research interests include nonlinear/stochastic/heuristic optimization, operations research, modelling and simulation, data analysis, and wireless sensor network. He has published in journals such as Computational Optimization and Applications, J. Optimization and Engineering, Journal of Optimization Theory and Applications, European Journal of Operational Research, Journal of Industrial and Management Optimization, Journal Applied Mathematics and Computation, etc. Mohamed Tawhid published more than 40 referred papers and edited 5 special issues in J. Optimization and Engineering (Springer), J. Abstract and Applied Analysis, J. Advanced Modeling and Optimization, and International Journal of Distributed Sensor Networks. Also, he has served on editorial board several journals. Also, he has worked on several industrial projects in BC, Canada.

Kurtis Manke is an undergraduate Bachelor of Science student, majoring in physics, at Thompson Rivers University (TRU). He has enjoyed a co-op work term with the DEAP-3600 team at SNOLAB (ON) (Canada), as well as several research assistant positions at TRU. His main research interests include: antenna theory, acoustics, high energy particle physics, medical physics, and applied mathematics.